Placement and Routing For A Field Programmable Multi-Chip Module

Sanko Lan Avi Ziv Abbas El Gamal

Information Systems Laboratory, Stanford University, Stanford, CA 94305

Abstract — Placement and routing heuristics for a Field Programmable Multi-Chip Module (FPMCM) are presented. The placement is done in three phases; partitioning, chip assignment and iterative improvement. The routing is done in two phases; global routing followed by detailed routing. Detailed routing involves new channel routing problems denoted by Exact Segmented Channel Routing (ESCR) and K-ESCR. A very fast K-ESCR heuristic is described. Experimental results show that the placement heuristic achieves high gate utilization, and that the K-ESCR heuristic performs surprisingly well over wide range of channel sizes.

I. Introduction

Using multiple reconfigurable Field-Programmable Gate Arrays (FPGAs) for logic emulation and rapid prototyping is becoming increasingly popular [9]. Different approaches for combining multiple FPGAs on a substrate have been proposed. First generation emulation machines employed an array of FPGAs mounted on printed circuit boards and, a fixed wiring network among the FPGAs. Inter-chip routing is done using both the fixed wires as well as the FPGAs themselves. The use of FPGAs for routing results in low FPGA utilization and poor performance. To improve FPGA utilization and performance, newer generations of emulation machines use a combination of FPGAs and dedicated routing chips interconnected via a fixed wiring network [2]. Inter-chip routing is done using the routing chips and the fixed wires.

Recently Dobbelaere *el al.* [1] proposed an alternative approach called a *Field Programmable Multi-Chip Module (FPMCM)* which integrates the logic of an FPGA with the fast interconnection of a routing chip. This is done by surrounding the logic core of an FPGA with a programmable interconnection frame. The frame supports fast through-chip routing as well as connections into the logic core. By using an MCM substrate with flip-chip bonding instead of a conventional PCB, a higher pin-to-gate ratio can be supported. As a result, an order of magnitude higher gate density and performance can be achieved using the proposed FPMCM than existing emulation machines.

To test the proposed FPMCM architecture we are developing an experimental CAD system for mapping large designs onto an FPMCM. The CAD system consists of two major modules, the MCM level module and the FPGA level module [5]. Since the proposed FPMCM can use an existing FPGA core and its accompanying FPGA CAD tools, we are focusing on the development of the MCM level tools. The MCM level tools accept as input an FPMCM description file which provides the details of the architecture of the FPMCM to be used, and a netlist for the design to be emulated. The system then places the design, i.e. finds an assignment of the design components into the chips, routes the inter-chip nets using both the fixed wiring network and the interconnection frames, and generates the information needed to program the frame switches.

The work was partially supported by FBI contract J-FBI-89-101

In this paper we describe the MCM-level placement and routing system. The FPMCM placement problem is divided into three phases; partitioning followed by chip assignment and iterative improvement. The netlist is first partitioned among the available chips using a modified Fiduccia-Mattheyses partitioning heuristic [3]. The partitions are then assigned to the FPGAs on the FPMCM so as to minimize routing cost. In the iterative improvement phase, components from different chips are moved around to improve a cost function of the routing.

Routing is invoked after placement has been completed, and is divided into two phases, *global routing* followed by *detailed routing.* The global router replaces each net by a set of horizontal or vertical two-point *connections.* Global routing of a single net is modeled as a Steiner tree problem [10]. To solve it we use a heuristic that combines a known heuristic for the general Steiner Tree Problem [8], with improvements that exploit the grid structure of the FPMCM.

To complete the routing, each connection must be assigned to one or more of the fixed wires. This task is broken into independent assignments for horizontal and vertical routing channels. We denote the problem of assigning connections to fixed wires in a channel by *Exact Segmented Channel Routing (ESCR).* The ESCR problem is similar to that of segmented channel routing [7], with a number of key differences that prevent us from using the heuristics reported in [7]. We describe a greedy ESCR algorithm that is simple and fast but gives surprisingly good results.

It is important to note that the placement and routing described in this paper, although developed specifically for FPMCM, can be easily adapted to any multi-FPGA connected by a fixed interconnect network.

The rest of the paper is organized as follows. In Section II a detailed description of the FPMCM routing architecture is given. Section III presents the two phases of the placement. In Section IV we present heuristics for global routing and detailed routing, and formally define the ESCR problem. Experimental results using an ESCR heuristic are also given. In Section V placement and routing results for benchmark designs mapped into two experimental FPMCMs are presented.

II. Routing Architecture

The FPMCM consists of an array of modified FPGAs mounted on a substrate and interconnected by a fixed wiring network. The logic core of the modified FPGA is assumed to be an SRAM-based FPGA core, although specialized functions such as memories, processors, etc., may also be used. The core is surrounded by an SRAM-programmable interconnection frame.

The I/O terminals of the FPGA core are connected to the programmable interconnection frame via *core pins*. The I/O terminals of the programmable interconnection frame are called *frame pins*. Frame pins on different chips are connected to each other via *frame wires* in the fixed wiring network. Other frame pins may be connected to external pins. The interconnection frame may be programmed to interconnect pairs of frame pins, resulting in fast through-chip interconnections.

As shown in Figure 1, the interconnection frame comprises four *switch boxes* placed at the corner regions of the chip. The switch boxes are SRAM programmable and are assumed to have complete flexibility, i.e., any horizontal frame wire may be connected to any vertical frame wire. Permutation boxes are placed between switch boxes. Again the permutation boxes are assumed to have complete flexibility; any two frame wires entering the box from opposite sides may be interconnected. Interconnections between the frame and the core pins are provided so that any signal entering a chip may be routed to the core and any signal leaving the core may be routed to a frame pin.

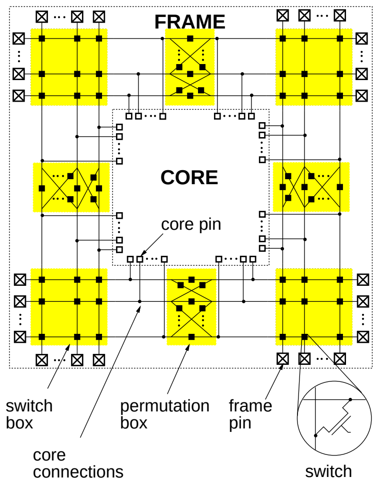

Figure 1: Programmable interconnection frame structure.

To simplify the routing problem as well as to decrease the average wire length, we restrict the fixed wiring network to consist of two-terminal horizontal or vertical fixed wires only. We also assume the pattern of the fixed wires terminating at any chip is independent of the location of the chip. At the edges of the array of chips this is done by "wrapping around" the fixed wires.

There are two types of paths through the interconnection frame, *I-paths* and *L-paths* as illustrated in Figure 2. Three types of L-paths are possible. Between the chip and its neighbor, the near-far type wastes two additional frame pins, while the far-far type wastes four.

III. Placement

The goal of the MCM level placement is to assign all components of the design to the chips without violating chip gate or pin capacities and such that all inter-chip nets can be routed using the available routing resources. To achieve MCM level

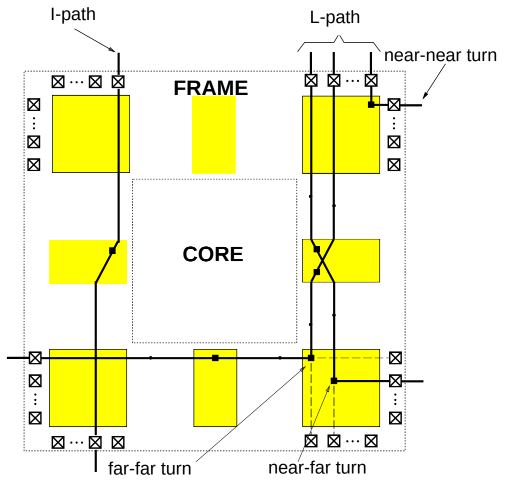

Figure 2: Making turns in the interconnection frame.

routability the placement attempts to minimize the number of connections to be routed. This is done by minimizing the number of inter-chip nets, reducing the number of chips each inter-chip net has to connect, and discouraging inter-chip nets from connecting chips that are not either on the same row or on the same column.

The placement is done in three phases; partitioning, chip assignment and iterative improvement. In the partitioning phase, we use a modified Fiduccia-Mattheyses partitioning heuristic [3] to minimize the number of inter-chip nets. In the chip assignment phase, we collapse components in the same partition into a node and merge nets that go to the same set of partitions into a weighted edge. Since our data show that only 1% of nets eventually go to more than 2 chips, hyperedges are removed for simplicity. We then assign the nodes to the chips on the MCM.

We start with a random placement and improve the placement using a simulated annealing heuristic [4]. The total routing cost is the sum of the routing cost for each edge. The routing cost of each edge is the weight of the edge multiplied by the cost of routing a net between the corresponding pair of chips. If the horizontal and vertical distances between the two chips are Δx and Δy respectively, the cost of routing a net between them is the sum of $\texttt{Net2Wire}(\Delta x)$ and $\texttt{Net2Wire}(\Delta y)$, where the function $\texttt{Net2Wire}(\cdot)$ computes an estimate of the number of frame wires that are needed to route a connection, which is a number between 1 and 2 because of our restriction on the frame wires that each connection is allowed to use.

After the chip assignment is completed, we improve the placement using a simulated annealing-based iterative improvement heuristic. The set of components considered at any iteration are those connected to at least one inter-chip net, which we denote by *frontier set*. At each iteration, we randomly pick a component in the frontier set, i.e. a *frontier component*, select a new destination chip for it, and evaluate the routing cost function. The acceptance or rejection of the move is then determined using the acceptance probability. The routing cost is again calculated using the $\texttt{Net2Wire}(\cdot)$ function, that gives a rough estimate of the number of frame wires needed to route the net.

IV. Routing

Routing in the FPMCM is done in two phases, global routing followed by detailed routing. Global routing is performed after placement is completed; thus it is assumed that each component in the design is assigned to a chip on the MCM. The global router replaces each net by a set of two-terminal *connections* and assigns each connection to a horizontal or vertical channel. As shown in Figure 3, there are two channels per row or column of chips. In the detailed routing phase the connections are assigned to specific fixed wires in the channels.

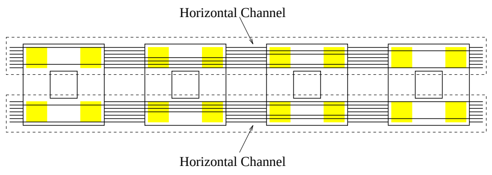

Figure 3: Two horizontal channels on the same row of chips

A. Global Routing

Global routing using frame wires is done one net at a time. We pick an inter-chip net at random, route it, and update the routing resources and costs.

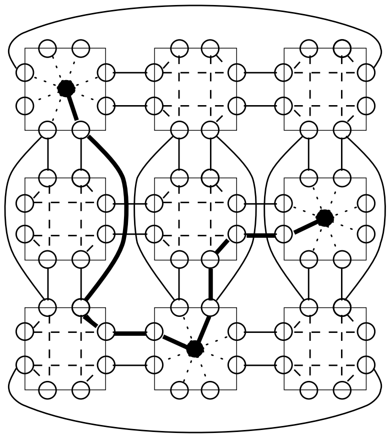

Figure 4: Modeling the routing problem as a Steiner tree problem. The squares denote chips and the empty circles denote nodes representing groups of frame pins. Full circles denote nodes representing groups of core pins.

Global routing of a single net can be formulated as a *Steiner Tree Problem (STP)* [10]. Each chip is represented by eight *frame nodes*, one for each of the eight groups of frame pins located at the outer two sides of the four switch boxes. Chips connected by the net have an additional *core node* located at the center, representing the core pins to one of which the signal must be connected. Figure 4 shows a 3×3 grid of chips with 3 core nodes at coordinates $(1,1)$, $(2,3)$ and $(3,2)$.

The edges of the graph represent the possible interconnections of frame nodes or core nodes to frame nodes. Associated with each edge is its cost. Since routing resources including frame pins and fixed wires are limited, the edge cost is a function of the frame pins used and the availability of frame wires. however, only estimates of these resources are available during global routing. There are three types of edges in the graph:

- **Internal edges.** If a net has a terminal at a particular chip, it must be connected to the core of that chip through a frame-to-core connection. Such a connection is represented by an internal edge. An internal edge is assigned a unit cost since it uses a single frame pin. In Figure 4 internal edges are shown as dotted lines.
- **Switching edges.** A chip that is not connected by the net may still be used to connect fixed wires in the fixed wiring network. The types of interconnections possible include switching edges for L-paths and I-paths. The cost assigned to a switching edge is 2, since in making such an interconnection we must use two frame pins. In Figure 4 switching edges are shown as dashed lines.

  Switching edges represent all types of turns in the frame as a combination of I-path edges and L-path edges. For example, a far-far turn which costs 6 frame pins is implemented with two I-path edges and an L-path edge. Since making a turn costs 2, 4, or 6 frame pins, it is correspondingly discouraged.
- **Routing edges.** These edges represent all possible horizontal and vertical interconnections among chips. Figure 4 shows a subset of the routing edges as solid lines.

  The cost given to a routing edge is determined by two factors: the length of the edge and an estimate of the availability of frame wires for realizing the edge. Because the global router does not assign fixed wires to connections, it can only estimate the contributions of these factors to cost.

Routing a net with minimal cost is equivalent to solving the Steiner tree problem on the graph described above, with the net terminals at core nodes. Figure 4 shows the graph as well as a global routing, in bold lines.

The heuristic we use to route a net consists of two phases. In the first phase, we use Takahashi and Matsuyama's [8] heuristic. In the second phase, the result is improved by repeated *corner flipping,* Π *to T conversion* and *leaf chip adjustment* until no more improvement is possible. Examples of these improvements are given in Figure 5.

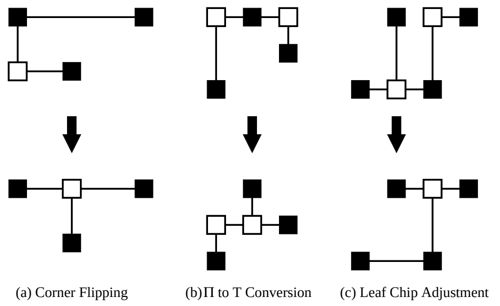

Figure 5: Single net grid improvements. The squares denote chips and the lines connections. Filled-in squares contain pins of the net.

B. Detailed Routing

To complete the routing, each connection must be assigned to one or more frame wires. This assignment is performed by the detailed router. As a result of the complete flexibility of the switch and permutation boxes and also assuming complete flexibility in core pin assignments it is easy to see that detailed routing reduces to independent *Exact Segmented Channel Routing* performed for all horizontal and vertical channels independently The inputs to ESCR are the *channel structure* and the *channel connections.*

Definition 1 *The* channel structure *is a set of frame wires* $\mathcal{F} = \{(F_1^s, F_1^t), (F_2^s, F_2^t), \ldots\}$, *where frame wire* F_i *connects the right (or bottom) side of chip* F_i^s *to the left (or top) side of chip* F_i^t, *respectively.*

Definition 2 *The* channel connections *are a set of connections* $\mathcal{C} = \{(C_1^s, C_1^t), (C_2^s, C_2^t), \ldots\}$ *where connection* C_i *connects the right (or bottom) side of chip* C_i^s *to the left (or top) side of chip* C_i^t, *respectively.*

Note that connection (i, j) is different from (j, i) in a wrap around channel structure because they connect different pairs of frame nodes. We now formally define the *Exact Segmented Channel Routing (ESCR)* problem.

Problem 1 *The* Exact Segmented Channel Routing *problem is to cover the channel connections* $\mathcal{C}$ *with disjoint subsets of the set of frame wires* $\mathcal{F}$, *such that a valid cover of* C_i *by* $\{F_{i_1}, F_{i_2}, \ldots, F_{i_k}\}$ *is one where* $C_i^s = F_{i_1}^s, F_{i_1}^t = F_{i_2}^s, \ldots, F_{i_{k-1}}^t = F_{i_k}^s, F_{i_k}^t = C_i^t$, *and the spans of the* F_{i_j} *do not intersect.*

In Problem 1, the number of frame wires used per connection is unlimited. This may result in unacceptable delays. By restricting the maximum number of frame wires allowed to cover a connection to an integer K, delays may be better controlled. We denote this version of the ESCR problem as the *K-wire Exact Segmented Channel Routing (K-ESCR)* problem.

Problem 2 *The* K-wire Exact Segmented Channel Routing *problem is an ESCR problem with the additional constraint that at most* K *frame wires can be used to cover each connection.*

The ESCR problem has been proved to be strongly NP-Complete [6]. In [6] Lan and How also proved that the general K-ESCR problem is strongly NP-complete when $K \geq 3$, and that a variation that prohibits track-permutation is strongly NP-complete for $K \geq 2$. The complexity of 2-ESCR is still an open problem.

However, in practice, the number of chips in a channel is bounded by a small constant (on the order of 16) which in turn imposes an upper bound on K. The ESCR with bounded channel length can be solved in polynomial time [6], but is too time-consuming in practice.

C. ESCR Heuristic

The heuristic we use to solve the ESCR problem routes each connection using no more than two frame wires, *i.e.,* $K = 2$. The algorithm consists of a greedy constructive phase followed by several iterations of a reroute phase; once a connection is routed, the reroute phase may alter this routing, but will never unroute the connection. In each successive reroute iteration, the heuristic searches more and more thoroughly for possible routings for the fewer and fewer remaining unrouted connections.

Since frame wires between the same pair of frame nodes are interchangeable in ESCR, we treat them as a single wire bundle. The capacity of a wire bundle is denoted by *Cap*(`wire bundle`). Similarly, a connection group consists of connections between the same pair of frame nodes. With the constraint of at most two frame wires per connection, a connection of length l can be routed in l different ways.

Assume the probabilities of routing the connection in each of these l ways are the same. Then, for each wire bundle, we can compute the expected number of connections that will use this wire bundle, denoted by *Exp*(`wire bundle`). We define the cost of using a frame wire in a wire bundle to be *Exp*(`wire bundle`) - *Cap*(`wire bundle`), unless *Cap* is zero in which case the cost is defined to be infinity. In addition, we define the cost of routing a connection to be the maximum of the costs of the wire bundles used.

In the greedy constructive phase, we attempt to route as many connections as possible, one by one. To route a connection of length l, we compute the cost of each of its l possible routings. If a least finite cost routing exists, we select it as the routing for the connection and update *Cap* and *Exp*. Otherwise, we skip the connection and leave it for the reroute iterations.

Example: Consider an ESCR problem with connections $(C_1^s, C_1^t) = (0, 3)$, $(C_2^s, C_2^t) = (1, 2)$, $(C_3^s, C_3^t) = (2, 0)$ and $(C_4^s, C_4^t) = (1, 0)$ and frame wires $(F_1^s, F_1^t) = (0, 1)$, $(F_2^s, F_2^t) = (1, 2)$, $(F_3^s, F_3^t) = (2, 3)$, $(F_4^s, F_4^t) = (3, 0)$, $(F_5^s, F_5^t) = (0, 2)$, $(F_6^s, F_6^t) = (2, 0)$, $(F_7^s, F_7^t) = (1, 3)$ and $(F_8^s, F_8^t) = (3, 1)$, as shown in Figure 6. The initial values of *Exp*, *Cap* and *Cost* are shown in Table I as Exp_0, Cap_0, and $Cost_0$. As an example of the calculation of these numbers, consider the value of *Exp* for F_2, which is $\frac{3}{2}$ because C_2 will use F_2 with probability 1 and C_4 will use F_2 with probability $\frac{1}{2}$.

To route C_1, we can use either F_1 and F_7 or F_3 and F_5. The costs for either alternatives is zero, so the router may pick either one; here we choose F_1 and F_7 and show the updated values in Table I as Exp_1, Cap_1, and $Cost_1$. To route C_2, the router can only use F_2; the updated values are shown in Table I as Exp_2, Cap_2, and $Cost_2$. Finally, to route C_3, we can use either F_3 and F_4 or just F_6. Once again both costs are zero, so the router may pick either one; here we choose the first again and show the updated values in Table I as Exp_3, Cap_3, and $Cost_3$. Since the costs of using F_2 and F_6 or F_4 and F_7 for C_4 are both infinity, C_4 will not be routable. Consequently C_4 must be left for the reroute phase to complete. ∎

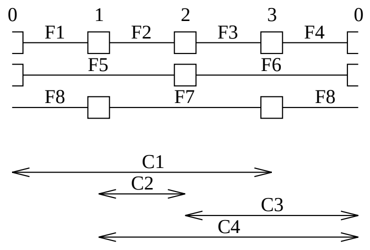

Figure 6: ESCR example.

Let L be the channel length. Computing *Exp* for all the wire bundles requires $O(L^3)$ time, because we have $O(L^2)$ connection groups and $O(L)$ time is required to compute the contribution of each connection group to *Exp*. After *Exp* is computed, for each connection we need $O(L)$ time to decide how to route this connection and $O(L)$ time to update *Exp* and *Cap*. Since L is a constant, the overall computation time is $O(n)$, where n is the number of connections.

The I^{th} reroute iteration tries to route each unrouted connection by rerouting up to $2I$ other connections. Thus from

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
C_1	$\frac{1}{2}$		$\frac{1}{2}$		$\frac{1}{2}$		$\frac{1}{2}$	
C_2		1						
C_3			$\frac{1}{2}$	$\frac{1}{2}$		$\frac{1}{2}$		
C_4		$\frac{1}{2}$		$\frac{1}{2}$		$\frac{1}{2}$	$\frac{1}{2}$	
Exp_0	$\frac{1}{2}$	$\frac{3}{2}$	1	1	$\frac{1}{2}$	1	1	0
Cap_0	1	1	1	1	1	1	1	1
$Cost_0$	$-\frac{1}{2}$	$\frac{1}{2}$	0	0	$-\frac{1}{2}$	0	0	-1
Exp_1	0	$\frac{3}{2}$	$\frac{1}{2}$	1	0	1	$\frac{1}{2}$	0
Cap_1	0	1	1	1	1	1	0	1
$Cost_1$	∞	$\frac{1}{2}$	$-\frac{1}{2}$	0	-1	0	∞	-1
Exp_2	0	$\frac{1}{2}$	$\frac{1}{2}$	1	0	1	$\frac{1}{2}$	0
Cap_2	0	0	1	1	1	1	0	1
$Cost_2$	∞	∞	$-\frac{1}{2}$	0	-1	0	∞	-1
Exp_3	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0
Cap_3	0	0	0	0	1	1	0	1
$Cost_3$	∞	∞	∞	∞	-1	$-\frac{1}{2}$	∞	-1

Table I: Initial values for Exp, Cap and $Cost$, and their values after $C_1 - C_3$ are routed.

one reroute iteration to the next, we increase the solution search depth. The algorithm is given in Figure 7.

Example: Consider again the ESCR problem depicted in Figure 6, and let $I = 2$. C_4 remains the only unrouted connection. It has two possible routings, using either F_2 and F_6, or F_7 and F_4.

`Avail(`F_2`)` will be called first to see whether F_2 may be made available by rerouting other connections. The answer is obviously *no,* because C_2 which is using F_2 cannot be rerouted. Thus the first alternative for routing C_4 fails.

The router will then try to see whether F_7 and F_4 are both available. It will first call `Avail(`F_7`)` which will determine whether F_7 can be released by rerouting C_1. Rerouting C_1 requires F_5, which is obviously available, as well as F_3, which is currently occupied by C_3. Therefore the answer from `AvailByRerouteNet(`C_1`,1,2)` depends on whether or not F_3 can be freed from implementing C_3. Since C_3 can be routed using F_6 which is currently available, `Avail(`F_3`)` will return *yes.* Thus the router concludes that F_7 is available by first rerouting C_3 using F_6 and then rerouting C_1 using F_5 and F_3.

The router will then check whether F_4 is available. Since F_4 was freed when C_3 was rerouted, `Avail(`F_4`)` will return *yes.* Thus C_4 can be routed with F_7 and F_4, and the router has found a solution for this example. ▮

By storing the values of the `Avail` function, we can avoid repeated determinations of whether a wire bundle is available. Since there are at most L^2 wire bundles, we will perform `Avail` at most L^2 times. In the worst case, each `Avail` calls `AvailByRerouteNet` $O(L)$ times, where each such call will need at most $O(L)$ trivial determinations of whether a wire bundle is available. Consequently we require $O(L^4)$ time to determine if other connections can be rerouted to route a particular unrouted connection, and the overall time required is $O(L^4 n) = O(n)$, where n is the number of connections.

This heuristic can be extended to the general K-ESCR problem. With bounded K and channel length, the complexity is still linear in the number of connections. If we relax the bound on channel length, the order of this heuristic increases with K.

```
RerouteUpto(I) {
  For each unrouted connection
    For each routing (s1,s2)
      If (Avail(s1,0,I) && Avail(s2,0,I)) {
        Reroute routed connections
        Route unrouted connection
      }
}

Avail(bundle,Dep,I) {
  If (bundle has unused) Return yes
  If (Dep ≥ I) Return no
  For each group using this bundle
    If (AvailByRerouteNet(group,Dep+1,I)) {
      Record the group
      Return yes
    }
  Return no
}

AvailByRerouteNet(group,Dep,I) {
  For each routing (s1,s2) of group
    If (Avail(s1,Dep,I) && Avail(s2,Dep,I)) {
      Record how to reroute
      Return yes
    }
  Return no
}
```

Figure 7: ESCR reroute iterations.

D. Results

In order to determine the quality of the proposed heuristic for the ESCR problem, we measured its performance on instances of randomly generated channel connections. The heuristic was executed for a channel with 8 chips and 50 pins on each side of each chip. The length of the frame wires were from 1 to 4. The number of frame wires per chip of each length are 20, 15, 10, and 5 respectively. The connections are randomly generated with the left edge of each connection uniformly distributed over the chips and the length of each connection independently distributed according to a geometric distribution.

The heuristic is executed for different *connection densities*, defined as the ratio between the number of connections and the number of frame wires. For densities of less than 0.55 the heuristic succeeded every time. When the density is above 0.6 the success ratio drops sharply. The heuristic achieves high *channel utilization*, defined as the ratio between the number of frame wires used for the routing and the overall number of frame wires. For the maximum connection density with success probability of 1 the average channel utilization is 0.8. The average running time on a SPARC IPX workstation for successful routing is less than 30ms for densities of 0.6 or less. The average running time for failed attempts is less than 100ms for the densities between 0.55 and 0.6. Similar results are achieved for channels of different lengths and widths.

V. Placement and Routing Results

We tested the MCM level placement and routing system, using several designs from the Partitioning93 benchmarks [11]. We placed and routed the designs on two experimental FPMCMs. FPMCM9 has 9 chips, in a 3×3 configuration; while FPMCM25 has 25 chips, in a 5×5 configuration. Table II gives the main parameters of both FPMCMs. All designs were successfully placed and routed on both FPMCMs. Tables IV–V describe the placement and routing results for the S38584 design, the largest design in the Partitioning93 benchmarks. The parameters of the design are given in Table III.

	FPMCM9	FPMCM25
Chips on MCM	9	25
Core size (gates)	4000	1600
Number of core pins	360	320
Number of frame pins	360	320

Table II: FPMCM parameters

Number of gates	33353
Number of components	22449
Number of nets	20719

Table III: S38584 design parameters

Table IV summarizes the results after placement and both phases of the routing. Table V presents the distribution of the number of connections used by the nets in the design. The figure shows that about 97% of the nets in the design are local nets, and do not use MCM resources for routing. Only less than 1% of the nets use more than 2 connections.

	FPMCM9	FPMCM25
Total inter-chip nets	601	624
Avg. core pins per chip	152	61
Max. core pins per chip	229	143
Number of connections	1189	1574
Avg. connections per inter-chip net	1.97	2.52
Avg. fixed wires per connection	1.00	1.13
Avg. frame pins per chip	264	144
Max. frame pins per chip	294	186

Table IV: Placement and routing results for S38584

Connections used	FPMCM9	FPMCM25
0	97.10%	96.99%
1	0.81%	0.14%
2	1.74%	2.34%
3	0.12%	0.11%
> 3	0.23%	0.43%

Table V: Distribution of connections used by nets in S38584

VI. Conclusions

A placement and routing system for the FPMCM proposed in [1] has been described. We demonstrated that using our system an FPMCM can implement designs of sizes at least one order of magnitude higher than those of single FPGAs with the same core utilizations. The placement and routing system can also be easily adapted to any multi-FPGA system with a fixed wiring network by routing directly through the FPGAs.

The results of mapping large Partition93 benchmark designs show that only 3% of the nets are inter-chip nets and less than 1% of the nets need more than 2 connections.

We formally defined the ESCR and K-ESCR problem. Heuristics for both global routing of a single net and the K-ESCR problem were presented. We demonstrated experimentally that the K-ESCR heuristic performs surprisingly well.

We plan to perform logic replication [3] after placement to further reduce the number of inter-chip nets.

Acknowledgments

The authors would like to thank Dana How for setting up a software development environment for this project and his valuable comments on the manuscript, and Jim Hwang for providing the partitioner source code and the netlist parser.

References

[1] I. Dobbelaere, A. El Gamal, D. How and B. Kleveland, “Field Programmable MCM Systems – Design of an Interconnection Frame,” *CICC '92*, Boston, MA, May 1992, pp. 4.6.1–4.6.4.

[2] R. Guo *et al.*, “A 1024 Pin Universal Interconnect Array with Routing Architecture,” *CICC '92*, Boston, MA, 1992, pp. 4.5.1–4.5.4.

[3] J. Hwang and A. El Gamal, “Min-Cut Replication in Partitioned Networks,” *IEEE Trans. on CAD*, in press.

[4] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, Vol. 220, May 1983, pp. 671–680.

[5] S. Lan, “Partitioning, Placement, and Routing for FPMCM,” Class Report for Integrated Systems Laboratory, Stanford University, May 1991.

[6] S. Lan, and D. L. How, “Complexity of Exact Segmented Channel Routing,” unpublished.

[7] V. Roychowdhury, J. Greene and A. El Gamal, “Segmented Channel Routing,” *IEEE Transactions on Computer Aided Design 12*, pp. 79–95, January 1993.

[8] H. Takahashi and A. Matsuyama, “An Approximate Solution for The Steiner Problem in Graphs,” *Math Japonica 24*, pp 573–577, 1980.

[9] S. Walters, “Computer-Aided Prototyping for ASIC-Based Systems,” *IEEE Design and Test of Computers*, June 1991, pp. 4–10.

[10] P. Winter, “Steiner Problem in Networks: A Survey,” *Networks 17*, pp. 129–167, 1987.

[11] *ACM/SIGDA Benchmark Newsletter*, June 1993.